Parallel FEM Simulation of Crack Propagation – Challenges, Status, and Perspectives*

Bruce Carter[1], Chuin-Shan Chen[1], L. Paul Chew[2], Nikos Chrisochoides[3], Guang R. Gao[4], Gerd Heber[1], Antony R. Ingraffea[1], Roland Krause[5], Chris Myers[1], Demian Nave[3], Keshav Pingali[2], Paul Stodghill[2], Stephen Vavasis[2], and Paul A. Wawrzynek[1]

[1] Cornell Fracture Group, Rhodes Hall, Cornell University, Ithaca, NY 14853
{bcarter,dchen,heber,myers}@tc.cornell.edu, wash@stout.cfg.cornell.edu, ari1@cornell.edu
[2] CS Department, Upson Hall, Cornell University, Ithaca, NY 14853
{chew,pingali,stodghil,vavasis}@cs.cornell.edu
[3] CS Department, University of Notre Dame, Notre Dame, IN 46556
{nikos,dnave}@cse.nd.edu
[4] EECIS Department, University of Delaware, Newark, DE 19716
ggao@capsl.udel.edu
[5] Center for Comp. Mech., Washington University in Saint Louis, MO 63130
rokrau@pocus.wustl.edu

Abstract. Understanding how fractures develop in materials is crucial to many disciplines, e.g., aeronautical engineering, material sciences, and geophysics. Fast and accurate computer simulation of crack propagation in realistic 3D structures would be a valuable tool for engineers and scientists exploring the fracture process in materials. In the following, we will describe a next generation crack propagation simulation software that aims to make this potential a reality.

1 Introduction

Within the scope of this paper, it is sufficient to think about crack propagation as a dynamic process of creating new surfaces within a solid. During the simulation, crack growth causes changes in the geometry and, sometimes, in the topology of the model. Roughly speaking, with the tools in place before the start of this project, a typical fracture analysis at a resolution of 10^4 degrees of freedom, using boundary elements, would take about 100 hours on a state-of-the-art single processor workstation. The goal of this project is it to create a parallel environment which allows the same analysis to be done, using finite elements, in 1 hour at a resolution of 10^6 degrees of freedom. In order to attain this level of performance, our system will have two features that are not found in current fracture analysis systems:

* This work was supported by NSF grants CCR-9720211, EIA-9726388, ACI-9870687, and EIA-9972853.

Parallelism – Current trends in computer hardware suggest that in the near future, high-end engineering workstations will be 8- or 16-way SMP "nodes", and departmental computational servers will be built by combining a number of these nodes using a high-performance network switch. Furthermore, the performance of each processor in these nodes will continue to grow. This will happen not only because of faster clock speeds, but also because finer-grain parallelism will be exploited via multi-way (or superscalar) execution and multi-threading.

Adaptivity – Cracks are (hopefully) very small compared with the dimension of the structure, and their growth is very dynamic in nature. Because of this, it is impossible to know a priori how fine a discretization is required to accurately predict crack growth. While it is possible to over-refine the discretization, this is undesirable, as it tends to dramatically increase the required computational resources. A better approach is to *adaptively* choose the discretization refinement. The dynamic nature of crack growth and the need to do adaptive refinement make crack propagation simulation a highly irregular application. Exploiting parallelism and adaptivity presents us with three major research challenges,

- developing algorithms for parallel mesh generation for unstructured 3D meshes with automatic element size control and provably good element quality,
- implementing fast and robust parallel sparse solvers, and
- determining efficient schemes for automatic, hybrid h-p refinement.

To tackle the challenges of developing this system, we have assembled a multi-disciplinary and multi-institutional team that draws upon a wide-ranging pool of talent and the resources of 4 universities.

2 System Overview

Figure 1 gives an overview of a typical simulation. During pre-processing, a solid model is created, problem specific boundary conditions (displacements, tractions, etc.) are imposed, and flaws (cracks) are introduced. In the next step, a volume mesh is created, and (linear elasticity) equations for the displacements are formulated and solved. An error estimator determines whether the desired accuracy has been reached, or further iterations, after subsequent adaptation, are necessary. Finally, the results are fed back into a fracture analysis tool for post-processing and crack propagation.

Figure 1 presents the simulation loop of our system in its final and most advanced form. Currently, we have sequential and parallel implementations of the outer simulation loop (i.e., not the inner refinement loop) running with the following restrictions: right now, the parallel mesher can handle only polygonal (non-curved) boundaries. Curved boundaries can be handled by the sequential meshers, though (see section 3). We have not yet implemented unstructured h-refinement and adaptive p-refinement, although the parallel formulator can handle arbitrary p-order elements.

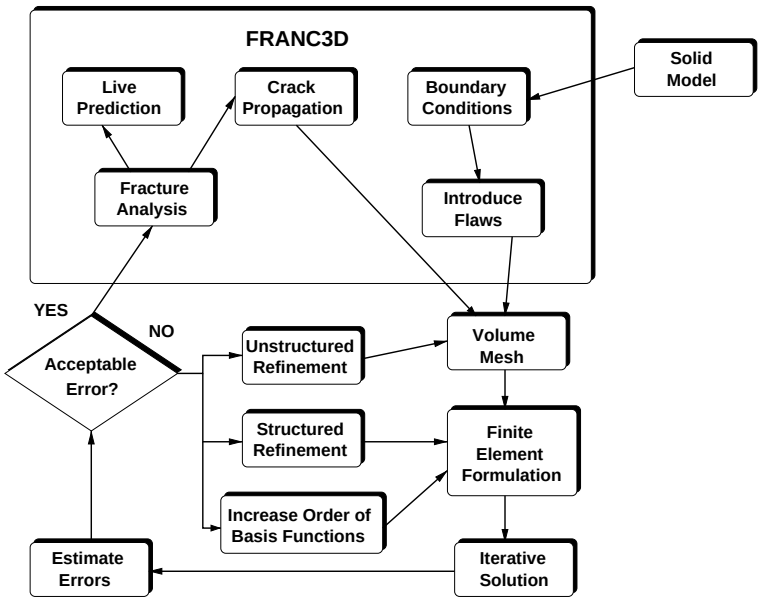

Fig. 1. Simulation loop.

3 Geometric Modeling and Mesh Generation

The solid modeler used in the project is called OSM. OSM, as well as the main pre- and post-processing tool, FRANC3D, is freely available from the Cornell Fracture Group's website [4]. FRANC3D - a workstation based FRacture ANalysis Code for simulating arbitrary non-planar 3D crack growth - has been under development since 1987, with hydraulic fracture and crack growth in aerospace structures as the primary application targets since its inception. While there are a few 3D fracture simulators available and a number of other software packages that can model cracks in 3D structures, these are severely limited by the crack geometries that they can represent (typically planar elliptical or semi-elliptical only). FRANC3D differs by providing a mechanism for representing the geometry and topology of 3D structures with arbitrary non-planar cracks, along with functions for 1) discretizing or meshing the structure, 2) attaching boundary conditions at the geometry level and allowing the mesh to inherit these values, and 3) modifying the geometry to allow crack growth but with only local remeshing required to complete the model. The simulation process is controlled by the user via a graphic user-interface, which includes windows for the display of the 3D structure and a menu/dialogue-box system for interacting with the program.

The creation of volume meshes for crack growth studies is quite challenging. The geometries tend to be complicated because of internal boundaries (cracks). The simulation requires smaller elements near each crack front in order to ac-

curately model high stresses and curved geometry. On the other hand, larger elements might be sufficient away from the crack front. There is a considerable difference between these two scales of element sizes, which amounts to three orders of magnitude in real life applications. A mesh generator must provide automatic element size control and give certain quality guarantees for elements. The mesh generators we studied so far are QMG by Steve Vavasis [14], JMESH by Joaquim Neto [11], and DMESH by Paul Chew [12]. These meshers represent three different approaches: octree-algorithm based (QMG), advancing front (JMESH), and Delaunay mesh (DMESH). QMG and DMESH come with quality guarantees for elements in terms of aspect ratio. All these mesh generators are sequential and give us insight into the generation of large "engineering quality" meshes. We decided to pursue the Delaunay mesh based approach first for a parallel implementation, which is described in [5]. *Departing from traditional approaches, we simultaneously do mesh generation and partitioning in parallel.* This not only eliminates most of the overhead of the traditional approach, it is almost a necessary condition to do crack growth simulations at this scale, where it is not always possible or too expensive to keep up with the geometry changes by doing structured h-refinement. The implementation is a parallelization of the so-called Bowyer-Watson (see the references in [5]) algorithm: given an initial Delaunay triangulation, we add a new point to the mesh, determine the simplex containing this point and the point's cavity (the union of simplices with non-empty circumspheres), and, finally, retriangulate this cavity. One of the challenges for a parallel implementation is that this cavity might extend across several submeshes (and processors). What looks like a problem, turns out to be the key element in unifying mesh generation and partitioning: the newly created elements, together with an adequate cost function, are the best candidates to do the "partitioning on the fly". We compared our results with Chaco and MeTis in terms of equidistribution of elements, relative quality of mesh separators, data migration, I/O, and total performance. Table 1 shows a runtime comparison between ParMeTis with PartGeomKway (PPGK) and, our implementation, called SMGP, on 16 processors of an IBM SP2 for meshes of up to 2000K elements. The numbers behind SMGP refer to different cost functions used in driving the partitioning [5].

Table 1. Total run time in seconds on 16 processors.

Mesh Size	PPGK	SMGP0	SMGP1	SMGP2	SMGP3
200K	90	42	42	42	42
500K	215	65	87	64	62
1000K	439	97	160	91	94
2000K	1232	133	310	110	135

4 Equation Solving and Preconditioning

So far, our focus has been on iterative solution methods. It is part of our future work to explore the potential of direct methods. We chose PETSc [13, 1] as the basis for our equation solver subsystem. PETSc provides a number of Krylov space solvers, such as Conjugate Gradient and GMRES, and a number of widely-used preconditioners, such as (S)SOR, and ILU/ICC. We have augmented the basic library with third party packages, including BlockSolve95 [8] and the Barnard's SPAI [2]. In addition, we have implemented a parallel version of the Global Extraction Element-By-Element (GEBE) preconditioner [7] (which is unrelated to the EBE preconditioner of Winget and Hughes [15]), and added it to the collection using PETSc's extension mechanisms. The central idea of GEBE is to extract subblocks of the global stiffness matrix associated with elements and invert them, which is highly parallel.

The ICC preconditioner is frequently used in practice, and is considered to be a good preconditioner for many elasticity problems. However, we were concerned that it would not scale well to the large number of processors required for our final system. We believed that GEBE would provide a more scalable implementation, and we hoped that it would converge nearly as well as ICC. In order to test our hypothesis, we ran several experiments on the Cornell Theory Center SP-2. The preliminary performance results for the `gear2` and `tee2` models are shown in Tables 2 and 3, respectively. `gear2` is a model of a power transmission gear with a crack in one of its teeth. `tee2` is a model of a T steel profile. For each model, we ran the Conjugant Gradient solver with both PETSc's IC(0) preconditioner and our own parallel implementation of GEBE on 8 to 64 processors. (**PC** – preconditioner type, p – number of processors, t_{PC} – time for preconditioner setup, t_{it} – time per cg iteration. The iteration counts in Tables 2 and 3 correspond to a 10^{15} reduction of the residual error, which is completely academic at this point.) The experimental results confirm our hypothesis: 1)

Table 2. Gear2 (79,656 unknowns)

PC	p	t_{PC}	t_{it}	Iters.
IC(0)	8	17.08	0.2	416
GEBE	8	9.43	0.19	487
IC(0)	16	15.47	0.27	422
GEBE	16	6.71	0.11	486
IC(0)	32	8.51	0.32	539
GEBE	32	3.73	0.08	485
IC(0)	64	11.00	0.28	417
GEBE	64	4.74	0.07	485

Table 3. Tee2 (319,994 unknowns)

PC	p	t_{PC}	t_{it}	Iters.
IC(0)	32	30.00	0.29	2109
GEBE	32	35.70	0.21	2421
IC(0)	64	23.60	0.29	2317
GEBE	64	7.60	0.12	2418

GEBE converges nearly as quickly as IC(0) for the problems that we tested.

2) Our naive GEBE implementation scales much better than PETSc's IC(0) implementation which uses BlockSolve95.

5 Adaptivity

Understanding the cost and impact of the different adaptivity options is the central point in our current activities. Our implementation follows the approach of Biswas and Oliker [3] and currently handles tetrahedra, while allowing enough flexibility for an extension to non-tetrahedral element types. For relatively simple, two-dimensional problems, stress intensity factors can be computed to an accuracy sufficient for engineering purposes with little mesh refinement by proper use of singularly enriched elements. There are many situations though when functionals other than stress intensity factors are of interest or when the singularity of the solution is not known *a priori*. In any case the engineer should be able to evaluate whether the data of interest have converged to some level of accuracy considered appropriate for the computation. It is generally sufficient to show, that the data of interest are converging sequences with respect to increasing degrees of freedom. Adaptive finite element methods are the most efficient way to achieve this goal and at the same time they are able to provide estimates of the remaining discretization error. We define the error of the finite element solution as $\mathbf{e} = \mathbf{u} - \mathbf{u}^{FE}$ and a possible measure for the discretization error is the energy norm, $||\mathbf{e}||^2_{E(\Omega)} = \frac{1}{2}\mathcal{B}(\mathbf{e}, \mathbf{e})$. The error estimator introduced by Kelly et.al. [9, 6] is derived by inserting the finite element solution into the original differential equation system and calculating a norm of the residual using interpolation estimates. An error indicator computable from local results of one element of the finite element solution is then derived and the corresponding error estimator is computed by summing the contribution of the error indicators over the entire domain. The error indicator is computed with a contribution from the interior residual of the element and a contribution of the *stress jumps* on the faces of an element. Details on the computation of the error estimator from the finite element solution can be found in [10].

6 Future Work

The main focus of our future work will be on improving the performance of the existing system. We have not yet done any specific performance tuning, like locality optimization. This is not only highly platform dependent, but also has to be put in perspective to the forthcoming runtime optimizations, like dynamic load balancing. We are considering introducing special elements at the crack tip, and non-tetrahedral elements (hexes, prisms, pyramids) elsewhere. On the solver side, we explore new preconditioners (e.g., support tree preconditioning), and multigrid, as well as sparse direct solvers, to make our environment more effective and robust. There is a port of our code base to the new 64 4-way SMP node NT cluster at the Cornell Theory Center underway.

7 Conclusions

At present, our project can claim two major contributions. The first is our parallel mesher/partitioner, which is the first practical implementation of its kind with quality guarantees. This technology makes it possible, for the first time, to fully automatically solve problems using unstructured h-refinement in a parallel setting. The second major contribution is to show that GEBE outperforms ICC, at least for our problem class. We have shown that, not only does GEBE converge almost as quickly as ICC, it is much more scalable in a parallel setting than ICC.

References

[1] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object-oriented numerical software libaries. In E. Arge, A.M. Bruaset, and H.P. Langtangen, editors, *Modern Software Tools in Scientific Computing*. Birkhauser Press, 1997.

[2] Stephen T. Barnard and Robert Clay. A portable MPI implementation of the SPAI preconditioner in ISIS++. In *Eighth SIAM Conference for Parallel Processing for Scientific Computing*, March 1997.

[3] R. Biswas and L. Oliker. A new procedure for dynamic adaption of three-dimensional unstructured grids. *Applied Numerical Mathematics*, 13:437–452, 1994.

[4] `http://www.cfg.cornell.edu/`.

[5] Nikos Chrisochoides and Demian Nave. Simultaneous mesh generation and partitioning for Delaunay meshes. In *8th Int'l. Meshing Roundtable*, 1999.

[6] J.P. de S.R. Gago, D.W. Kelly, O.C. Zienkiewicz, and I. Babuška. A posteriori error analysis and adaptive processes in the finite element method: Part II – Adaptive mesh refinement. *International Journal for Numerical Methods in Engineering*, 19:1621–1656, 1983.

[7] I. Hladik, M.B. Reed, and G. Swoboda. Robust preconditioners for linear elasticity FEM analyses. *International Journal for Numerical Methods in Engineering*, 40:2109–2127, 1997.

[8] Mark T. Jones and Paul E. Plassmann. Blocksolve95 users manual: Scalable library software for the parallel solution of sparse linear systems. Technical Report ANL-95/48, Argonne National Laboratory, December 1995.

[9] D.W. Kelly, J.P. de S.R. Gago, O.C. Zienkiewicz, and I. Babuška. A posteriori error analysis and adaptive processes in the finite element method: Part I – Error analysis. *International Journal for Numerical Methods in Engineering*, 19:1593–1619, 1983.

[10] Roland Krause. *Multiscale Computations with a Combined h- and p-Version of the Finite Element Method*. PhD thesis, Universität Dortmund, 1996.

[11] J.B.C. Neto et al. An algorithm for three-dimensional mesh generation for arbitrary regions with cracks. submitted for publication.

[12] `http://www.cs.cornell.edu/People/chew/chew.html`.

[13] `http://www.mcs.anl.gov/petsc/index.html`.

[14] `http://www.cs.cornell.edu/vavasis/vavasis.html`.

[15] J.M. Winget and T.J.R. Hughes. Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies. *Computational Methods in Applied Mechanical Engineering*, 52:711–815, 1985.